AMENDMENTS

In the Specification:

Please amend the paragraph on p. 1 spanning lines 11 through 17 as shown below:

This application is a continuation application claiming priority from U.S. Patent Application Serial No. 09/287,393, filed on April 4, 1999, entitled "Secure Execution of Program Code" and naming Butler W. Lampson and Paul England as inventors, the disclosure of which is incorporated herein by reference. This application is related to co-pending commonly assigned provisional application ~~attorney docket MS1-282US ("System and Method for Authenticating an Operating System and proving a CPU/OS to a Third Party")~~ Serial No. 60/105,891, filed on Oct. 26, 1998, entitled "System and Method for Authenticating an Operating System to a Central Processing Unit, Providing the CPU/OS With Secure Storage, and Authenticating the CPU/OS to a Third Party", application ~~docket 777.206US1~~ Serial No. 09/227,611, filed on Jan. 8, 1999, now Patent No. 6,327,652, entitled ~~(~~ "Loading and Identifying a Digital Rights Management Operating System " ~~)~~, application ~~docket 777.211US1 (~~ Serial No. 09/227,568, filed Jan. 8, 1999, entitled "Key-Based Secure Storage"~~)~~, and application ~~docket 777.212US1 (~~ Serial No. 09/227,559, filed Jan. 8, 1999, entitled "Digital Rights Management Using One Or More Access Prediates, Rights Manager Certificates, And Licenses"~~)~~. The disclosures of these applications are hereby incorporated by reference.

The paragraph spanning p. 3, line 27 through page 4, line 5 has been amended as shown below:

The fundamental building block for client-side content security is a secure operating system. If a computer can be booted into an operating system that is trusted to honor content rights, and only allows authorized applications to access rights-restricted data, then data integrity within the machine can be assured. The stepping-stone to a secure operating system is sometimes called "Secure Boot". If secure boot cannot be assured, whatever rights management system the OS provides can always be subverted by booting into an insecure operating system.

The paragraph spanning p. 4, lines 6 through 13 has been amended as shown below:

Secure boot of an operating system is usually a multi-stage process. A securely booted computer runs a trusted program at startup. The trusted program loads another program and checks its integrity, e.g., by using a code signature, before allowing it to run. This program in turn loads and checks subsequent layers. This proceeds all the way to loading trusted device drivers, and finally a trusted application. Related patent application ~~MS1-282US~~ Serial No. 60/105,891 describes an overall method of securely booting an operating system, and also notes related technology.

The paragraph spanning p. 7 line 23 through p. 8, line 16 has been amended as shown below:

Hardware components 120 are shown as a conventional personal computer (PC) including a number of components coupled together by one or more system buses 121 for carrying instructions, data, and control signals. These buses may assume a number of forms, such as the conventional ISA, PCI, and AGP buses. Some or all of the units coupled to a bus can act as a bus master for initiating transfers to other units. Processing unit 130 may have one or more microprocessors 131 driven by system clock 132 and coupled to one or more buses 121 by controllers 133. Internal memory ~~system140~~ system 140 supplies instructions and data to processing unit 130. High-speed RAM ~~141stores~~ 141 stores any or all of the elements of software 110. ROM 142 commonly stores basic input/output system (BIOS) software for starting PC 120 and for controlling low-level operations among its components. Bulk storage subsystem 150 stores one or more elements of software 110. Hard disk drive 151 stores software 110 in a nonvolatile form. Drives 152 read and write software on removable media such as magnetic diskette 153 and optical disc 154. Other technologies for bulk storage are also known in the art. Adapters 155 couple the storage devices to system buses 121, and sometimes to each other directly. Other hardware units and adapters, indicated generally at 160, may perform specialized functions such as data encryption, signal processing, and the like, under the control of the processor or another unit on the buses.

The paragraph spanning p. 8, lines 17 through 27 has been amended as shown below:

Input/output (I/O) subsystem 170 has a number of specialized adapters 171 for connecting PC 120 to external devices for interfacing with a user. A monitor 172 creates a visual display of graphic data in any of several known forms. Speakers ~~173~~ output audio data that may arrive at an adapter 171 as digital wave samples, musical-instrument digital interface (MIDI) streams, or other formats. Keyboard 174 accepts keystrokes from the user. A mouse or other pointing device 175 indicates where a user action is to occur. Block 176 represents other input and/or output devices, such as a small camera or microphone for converting video and audio input signals into digital data. Other input and output devices, such as printers and scanners commonly connect to standardized ports 177. These ports include parallel, serial, SCSI, USB, FireWire, and other conventional forms.

The paragraph on p. 9 spanning lines 1 through 14 has been amended as shown below:

Personal computers frequently connect to other computers in networks. For example, local area network (LAN) 180 ~~connect~~ connects PC 120 to other PCs 120' and/or to remote servers 181 through a network adapter 182 in PC 120, using a standard protocol such as Ethernet or token-ring. Although Fig. 1 shows a physical cable 183 for interconnecting the LAN, wireless, optical, and other technologies are also available. Other networks, such as wide-area network (WAN) 190 can also interconnect PCs 120 and 120', and even servers 181, to remote computers 191. Fig. 1 illustrates a communications facility 192 such as a public switched telephone network for a WAN 190 such as an intranet or the internet. PC 120 can employ an internal or external modem 193 coupled to serial port 177. Other technologies such as packet-switching ISDN, ATM, DSL, and frame-relay are also available. In a networked or distributed-computing environment, some of the software 110 may be stored on the other peer PCs 120', or on computers 181 and 191, each of which has its own storage devices and media.

The paragraph on p. 9 spanning lines 15 through 23 has been amended as shown below:

Software elements 110 may be divided into a number of types whose designations overlap to some degree. For example, the previously mentioned BIOS sometimes includes high-level routines or programs which might also be classified as part of an operating system (OS) in other settings. The major purpose of OS 111 is to provide a software environment for executing application programs 112 and for managing the resources of system 100. An OS such as the Microsoft® Windows® operating system or the Windows NT® operating system commonly implements high-level application-program interfaces (APIs), file systems, communications protocols, input/output data conversions, and other functions.

The paragraph spanning p. 11, lines 3 through 11 has been amended as shown below:

Secure ring 220, also called Ring B, is an inner ring to Ring C, and has full access privileges to its outer Ring C; but Ring B is in turn an outer ring with respect to ring A, and thus has only restricted access to this inner ring. In this embodiment, the major purpose of Ring B is to hold most of the code that carries out authenticated-boot operations as mentioned above and in Application docket ~~MS1-282US~~ Serial No. 60/105,891. Thus, it can have both semipermanent storage such as nonvolatile flash RAM for code routines and volatile read/write memory for temporary data such as keys. A megabyte or less of the total address range would likely suffice for Ring B.

The paragraph spanning p. 11, lines 12 through 27 has been amended as shown below:

Secure ring 230, also called Ring A is an inner ring to both Rings B and C, and has full access to them for both code and data. It can also employ both nonvolatile and volatile technologies for storing code and data respectively. Its purpose in this embodiment is to store short loader and verifier programs and keys for authentication and encryption. Under the proper conditions, this code and data can be loaded in the clear. The address space required by Ring A is generally much smaller than that of Ring B. That is, this exemplary embodiment has the Ring A address range within the address range of Ring B, which in turn lies within the address range of Ring C. The address ranges of the rings need not be contiguous or lie in a single block. In order to prevent the access restrictions of the curtained rings from being mapped away by a processor, the address ranges of Rings A and B can be treated as physical addresses only. In one embodiment, virtual addresses are conventionally translated into their corresponding real addresses, and then the restrictions are interposed at the level of the resulting real addresses. Alternatively, ~~a~~ one or more mechanisms could disable virtual addressing when certain addresses are accessed.

The paragraph spanning p. 13, lines 15 through 22 has been amended as shown below:

However, there are no restrictions on the code that can be loaded into any of the Ring-B memory areas. Examples of Ring-B code include smartcard-like applications for key management, secure storage, signing, and authentication. Further examples include electronic cash storage, a secure interpreter for executing encrypted code, and modules for providing a software licenses necessary for a piece of software to run. It is also possible to load only a part of an application, such as a module that communicates with a media player in unsecure memory for reducing software piracy.

The paragraph on p. 15 spanning lines 1 through 11 has been amended as shown below:

Control unit 350 further includes a specification or ~~map354~~ map 354 of one or more address ranges of the memory addresses desired to be curtained. The specification can be in any desired form, such as logic circuitry, a read-only table of addresses or extents, or even a small writable or rewritable storage array. If the addresses are in memories having separate address sequences, additional data specifying the particular memories can be added to the addresses within each sequence. A detector or comparator 355 receives the contents of instruction pointer 352 and the curtained-memory map 354. A curtained memory having multiple rings, subrings, or other levels can have a separate specification for each of the curtained regions. Alternatively, a single specification can explicitly designate the ring or subring that each address range in the specification belongs to.

The paragraph spanning p. 17, lines 4 through 13 has been amended as shown below:

Logic 356 determines whether or not to execute the code by comparing the privilege level of the calling code and the operation-index parameter, and potentially whether the processor is already executing some other curtained code, with entries in a jump-target table 357 stored in a location accessible to it. The logic to enforce these requirements can be implemented in the ~~memory controller~~ logic 356, or by code executing in a highly privileged ring such as Ring A. Table I below illustrates one form of jump-target table. The table can be stored in the same curtained memory block as the code itself, or in a memory block that is more privileged; or it can be stored in special-purpose storage internal to the CPU or memory manager.

The paragraph spanning p. 18, line 25 through p. 19, line 8 has been amended as shown below:

Interrupts offer almost unlimited access to system resources. A simple way to prevent an interrupt from subverting curtained code is to issue a privileged instruction that causes a microprocessor to switch off all interrupts until a companion instruction switches them back on. A new instruction such as SnoopIntrerrupts Ring, Subring, OpIndex can call a curtained operation instead of the requested interrupt routine when an interrupt tries to access memory in a designated ring or subring, or operation. ~~1This~~ This can also be managed by having the curtained code set up the interrupt handlers to execute trusted curtained code. However, it is still important ~~to protect~~ that the entry point into the curtained operation (that sets the interrupt vector) itself be protected against interruption so that the interrupt mechanism ~~can~~ cannot be subverted by a malicious program.

The paragraph spanning p. 19, lines 9 through 23 has been amended as shown below:

An instruction having the form SetOpaque MemoryStart, MemoryLength / SetInterruptThrowError / SetTransparent does not switch off interrupts, but rather modifies the ~~microprocessor''~~ microprocessor's behavior. When an interrupt occurs, the processor clears all registers, except the stack pointer, before the interrupt is fielded. It is useful for long-running curtained operations that could reveal sensitive information, such as partial keys, if they were interrupted. An operand of this instruction can specify a memory range that the processor also clears before the interrupt is serviced. The first switch of the instruction activates a variant that causes a processor fault when an interrupt occurs, even in user mode. The user code can then disable operations and process—or decide not to process—the interrupt. The second switch turns off the SetOpaque execution mode. These can be user-mode operations, if desired. In at least some circumstances, this instruction should fault the processor when returning from the interrupt, to prevent an undesired jump into the middle of curtained code that might have been executing when the interrupt took control.